Aula 9

Promela

- "PROcess MEtaLAnguage"desenvolvida para
- SPIN Model checker `spinroot.com`
- em 2002 obteve o ACM System Software Award
- Um programa $\bar{P}$ é um conjunto de processos

  $$P_1, \ldots P_n$$

  que executam concorrentemente com:
- partilha de variáveis
- canais FIFO ou síncronos
- para descrever cada processo usa-se uma *linguagem de comandos guardados*: instruções de linguagens imperativas, ações de comunicação e regiões atómicas
- comandos guardados: têm uma guarda e uma ação
- ver também [BA08]

Subconjunto de instruções

proctype *name*() {stmt}

$$
\begin{array}{rcl}
\mathsf{stmt} & ::= & \texttt{skip} \mid x := \mathsf{expr} \mid c?x \mid c!\mathsf{expr} \mid \\
& & \mathsf{stmt}_1 \,;\, \mathsf{stmt}_2 \mid \texttt{atomic}\{\mathsf{assignments}\} \mid \\
& & \textbf{if} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{fi} \quad \mid \\
& & \textbf{do} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{do}
\end{array}
$$

- as variáveis globais são declaradas fora dos processos
- as variáveis locais são declaradas nos processos
- g_i são condições sobre as variáveis
- *skip* comando que termina num passo e não altera nada
- ; ou $\Rightarrow$ execução sequencial
- `atomic` passo que não pode ser intercalado com outros
- ver semântica operacional em [BKL08] (Cap. 2.2.5).

Seleção if-fi

$$\textbf{if} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{fi}$$

- escolha não determinística dos $stmt_i$ para os quais *a guarda* g_i *se verifica* no estado actual.
- supõe-se que todo isso é executado num passo atómico.
- se nenhuma das guardas $g_1, \ldots, g_n$ se verificar no estado corrente o processo fica *bloqueado*.
- neste caso a execução de outros processos poderão desbloqueá-lo.
- notar que o **if-then-else** imperativo corresponde a

$$\textbf{if} \ :: g \Rightarrow \mathsf{stmt}_1 \ :: \neg g \Rightarrow \mathsf{stmt}_2 \ \textbf{fi},$$

- Mas se existir **::else** ele evita o bloqueio no caso de *todas* as guardas forem falsas.

Repetição do-od

$$\textbf{do} \quad :: g_1 \Rightarrow \mathsf{stmt}_1 \quad \ldots \quad :: g_n \Rightarrow \mathsf{stmt}_n \quad \textbf{do}$$

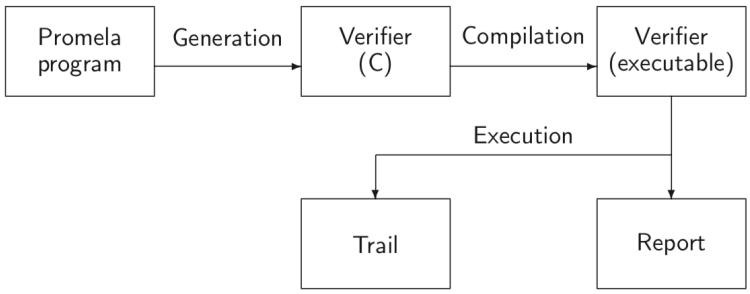
- Execução repetida da escolha não determinística entre os comandos guardados cujas guardas sejam verdadeiras
- se todas as guardas forem falsas não bloqueia mas a repetição termina e a execução passa para o comando seguinte.
- também pode terminar com o comando **break**
- ou ter um comando **::else**
- existe também a *instrução de salto* **goto** *label*, onde *label:* deve etiquetar a instrução que deverá ser executada em seguida.

Verificação de programas sequênciais

- Asserções que permitem verificar programas sequênciais
- **assert**(*cond*)
- um Model checker verifica todas as possíveis computações (não-determinísticas)
- e se *cond* não se verificar para alguma delas é retornado um contra-exemplo.
- na verificação por dedução não será feito assim...

Verificação com SPIN

- Para ser mais eficiente o SPIN gera um "verificador"em linguagem C que depois de compilado permite executar a verificação.
- `spin -a max.pml; gcc -o pan pan.c; ./pan`
- Para analisar erros: `spin -t max.pml`.



Exemplos

- max.pml
- mdc.pml
- semaforo.pml

Verificação de programas concorrentes

Quando há mais que um processo.

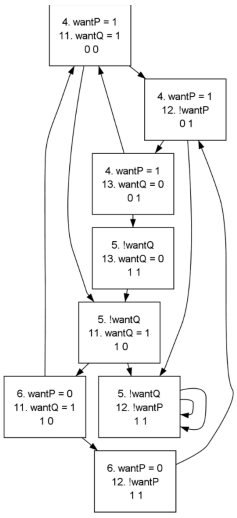
- Em Promela:
  - Executar várias cópias:**active** [n] P()
  - **_pid** indica qual o processo a correr
  - **_nr_pr** indica o número de processos activos
  - em vez de **active**
  - primeiro processo: **init** { ... }
  - usar **run** para executar um processo especifico. Ex: **run**(P(1,5))
  - retorna o pid do processo (não é uma instrução)
- no SPIN:
  - Execução aleatória:`spin pq.pml`

- Execução interactiva: `spin -i pq.pml`
- Construção de um verificador: `spin -a pq.pml`
- ... ver outras opções
- npr.pml , nprc.pml

Exclusão Mutua

- versão naive com contador me.pml
- com sincronização busy-waiting: me2.pml
- com deadlock: me1.pml
- com semáforo: mes.pml
- implementar também o algoritmo de Peterson (prática)

LTS para me1.pml



Canais em Promela

```
chan ch = [capacity] of { typename, ..., typename }
```

- que permite definir canais em que cada mensagem tem vários campos cada um com um dado tipo.
- se capacidade = 0 são síncronos

- *ch*!1 enviar 1
- *ch*?*x* receber um valor e guardar em *x*
- em geral são declarados globalmente
- se forem locais "desaparecem"se o processo em que foram criados terminar
- podem ser passados com argumentos de processos
- nas ações de receber mensagens a variável pode ser "anónima"_ caso só interesse saber se algo foi recebido.
- pode haver *arrays* de canais: **chan** [2] = [3] of {**byte**, **bool**}
- **full**, **nfull**, **empty**, **nempty** funções Booleanas que testam o estado dos canais
- Rendezvous
- Cliente-servidor : cs.pml a cs5.pml
- Buffers
- verificar se estão cheios ou vazios: csr2.pml

Referências

[BA08] Mordechai Ben-Ari. *Principles of the Spin Model Checker*. Springer, 2008.

[BKL08] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of Model Checking*. MIT Press, 2008.